JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 193-198
Wysłane: 2019-06-14
Przyjęte: 2019-06-22

Porównanie efektywności składowania modeli UML w wybranych technologiach bazodanowych

Andrii Filatov, Paweł Flis*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł odpowiada na pytanie, która baza danych jest najlepszym wyborem do efektywnego składowania danych opisujących modele UML. Wzięto pod uwagę trzy produkty: MongoDB, PostgreSQL i Neo4J. Na badanie efektywności składa się pomiar czasu odpowiedzi zapytań zapisujących oraz pobierających dane. Uwzględnia również stopień wzrostu pamięci podczas wstawiania danych oraz ocenę poziomu skomplikowania implementacji mapującej dane testowe do wykorzystania w zapytaniach do baz danych.

Słowa kluczowe:UML; MongoDB; Neo4J; PostreSQL

*Autor do korespondencji.

Adresy e-mail: andriymasteridze@gmail.com, stercage@gmail.com

Storage efficiency comparison of UML models in selected database technologies

Andrii Filatov, Paweł Flis*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B,20-618 Lublin, Poland

Abstract. The study answers the question which database is the best choice for efficient data storage of UML models. Three products were considered: MongoDB, PostgreSQL and Neo4J. The effectiveness test consists of measurement the response time of queries that save and load data. This study also take into account the memory increase ratio during data insertion and the level of complexity of the implementation of the test data mappers used in database queries.

Keywords: UML; MongoDB; Neo4J; PostgreSQL

*Corresponding author.

E-mail addresses: andriymasteridze@gmail.com, stercage@gmail.com

1. Wstęp

W obecnych czasach, modelowanie staje się coraz ważniejszym elementem wytwarzania oprogramowania, od którego w dużym stopniu zależy sukces całego przedsięwzięcia. Pozwala ono graficznie wyrazić najważniejsze cechy projektu oraz jego otoczenia. Jest mniej abstrakcyjne niż kod programu. W efekcie pozwala to na łatwe zaprojektowanie oprogramowania i częściowo automatyczną implementację jego kodu. Dzięki modelowaniu uzyskuje się ogólny obraz systemu. Można więc przedstawić abstrakcyjny model systemu klientowi przed jego powstaniem. W przypadku, gdy nastąpi potrzeba zmiany lub dodania nowej funkcjonalności, wystarczy zmienić to w modelu. Redukuje to koszt i czas realizacji całego projektu.

Na dzień dzisiejszy, 30-letnie języki i metody programowania są nadal używane. Zaistniała jednak pilna potrzeba nowoczesnego podejścia, aby uniknąć błędów programowania, które prowadzą do konieczności długotrwałego rozwiązywania problemów. Nowoczesne programowanie zorientowane na modelowanie (ang. Model Oriented Programming) nie wymaga kodowania ręcznego, kod tworzy komputer, więc wykluczone są wszelkie błędy występujące podczas ręcznej transkrypcji abstrakcyjnego modelu w kodzie oprogramowania[1].

Ponieważ stosowanie modeli staje się coraz bardziej pożądane, należałoby zadbać o właściwy sposób ich przechowywania, tak, aby dostęp do danych możliwy był w jak najkrótszym czasie i jak najniższym kosztem.

Ze względu na rozmiar modeli wykorzystanie pewnego typu bazy danych staje się wymogiem. Można wybierać spośród baz relacyjnych oraz nierelacyjnych (NoSQL). Porównując, bazy relacyjne z różnego typu bazami nierelacyjnymi, pod względem szybkości zapisu i odczytu danych, trudno wyłonić zwycięzcę [2, 3, 4]. Żadna z porównywanych baz nie wygrywa dla wszystkich przypadków testowych. Z tego powodu jednoznaczny wybór bazy danych odpowiedniej do przechowywania modeli nie jest łatwy.

Z wymienionych wcześniej powodów przeprowadzono badanie mające rozstrzygnąć opisany problem. Polegało ono na analizie porównawczej efektywności składowania oraz dostępu do modeli UML na przykładzie wybranych technologii bazodanowych. Efektywność została określona w kilku kategoriach, m. in. czasu dostępu do danych, czasu wstawiania danych, poziomu skomplikowania całego procesu oraz ilości zajmowanej pamięci. Dodatkowo cel badań obejmował utworzenie aplikacji, pozwalającej na przeprowadzenie badań, zebranie wyników oraz ich przetworzenie. Każda z wybranych technologii bazodanowych była reprezentowana przez odpowiednio:

PostgreSQL – bazy relacyjne, MongoDB – bazy dokumentowe, Noe4j – bazy grafowe.

Została postawiona następująca hipoteza badawcza: ***Baza dokumentowa reprezentowana przez MongoDB jest najlepszym wyborem w kwestii efektywnej obsługi struktur danych modeli UML (ang. Unified Modeling Language).***

2. Przegląd litaratury

Nie znaleziono badań zajmujących się dokładnie opisywanym problemem, dlatego przeanalizowano kilka o podobnej tematyce.

Analizę parametrów baz danych z punktu widzenia praktycznego wykorzystania z dużymi zbiorami danych przedstawiono w artykule napisanym przez grupę Marokańskich badaczy[5]. Porównanie zostało dokonane między bazami SQL oraz NoSQL. Zbadano ich bezpieczeństwo, skalowalność, wydajność itp. Wstęp nadmienia, że NoSQL wprowadzono i udoskonalono w celu obsługi Big Data, natomiast język SQL służy do obsługi strukturalnych baz danych i wykonywania transakcji. Bazy danych SQL są skalowalne tylko pionowo, więc aby poradzić sobie z rosnącym obciążeniem, występuje potrzeba zwiększenia pojemności i wydajności na serwerze. Aby to osiągnąć, można na przykład zwiększyć szybkość procesora, pojemność dysku SSD serwera bazy danych lub pamięć RAM. Jednak odkładanie wielu tabel w dużych klastrach lub siatkach jest kosztowne i złożone. Natomiast baza NoSQL jest skalowalna w poziomie. Zatem, obsługa dużych zbiorów danych polega na dodaniu serwera do struktury bazy danych, w wyniku czego, używanie NoSQL daje łatwą i tanią skalowalność systemu. Z drugiej strony relacyjne bazy danych wymagają predefiniowanego modelu danych i danych strukturalnych. Oferują one zaawansowaną funkcjonalność do zarządzania, aktualizacji i wyszukiwania danych za pomocą SQL.

Badanie na temat porównania wydajności baz danych dokonali badacze z Korei[2]. Opracowali program, który umożliwiał wstawianie, aktualizowanie, wybieranie i usuwanie w celu porównania baz relacyjnych (PostgreSQL) i nie relacyjnych (MongoDB). Te operacje zostały wykonane dla różnych rozmiarówdanych. A mianowicie dla każdego z 30000, 90000, 150000, 210000 i 300000. Eksperyment obejmował operację wstawiania, która została wykonana najpierw w PostgreSQL, następnie w MongoDB zaprojektowaną podobnie do modelu relacyjnego, a potem z projektem podobnym do modelu niestrukturalnego. Dla każdej operacji wstawiania porównano czas, jaki upłynął od wyboru, aż do dostępności danych. Badania wykazały, że dla każdej operacji zarówno z projektowaniem niestrukturalnym jak i projektowaniem z relacyjnym modelem danych, baza MongoDB była szybsza. Dodatkowo wykazano, że projektowanie z niestrukturalnym modelem danych wydaje się lepsze niż projektowanie z relacyjnym modelem danych. Używanie MongoDB z niestrukturalnym modelem danych zapewnia ogólną poprawę wydajności. Jednakże, gdy środowisko wymaga precyzyjnego i uporządkowanego modelu danych, używanie RDBMS (ang. Relational Database Management System), takich jak PostgreSQL, będzie wykazywać wyższą jakość wykonania.

Holenderscy autorzy artykułu[4] poszukiwali bazy danych, która by była wydajna podczas wykonywania pojedynczych zapisów i wielu odczytów. Porównano 3 bazy danych pod kątem wydajności. Są to MongoDB, Cassandra, PostgreSQL. Żadna z nich nie sprawdziła się w obu przypadkach. PostgreSQL wygrała przy wielu odczytach, natomiast MongoDB wygrała w kategorii pojedynczych zapisów.

Odpowiedź na pytanie, która baza danych SQL i NoSQL jest najlepsza dla dużych zbiorów danych zawiera artykuł [6]. Ponieważ wiele danych jest niestrukturalnych lub pół-strukturalnych, wynika potrzeba bazy danych, która byłaby w stanie sprawnie je przechowywać. W relacyjnych bazach danych niemożliwe jest szybkie dodanie nowych typów danych, które by nie pasowały do strukturalnych danych, ponieważ schemat jest sztywno zdefiniowany. Natomiast bazy NoSQL zapewniają taki model danych, który lepiej odwzorowuje te potrzeby.

Praca Douglas Kunda i HazaelPhiri z uniwersytetu Mulungushi[7] miała na celu odpowiedzieć na pytanie, czy bazy NoSQL zastąpią bazy relacyjne. Bazy NoSQL stają się coraz popularniejszym wyborem, ponieważ bardziej odpowiadają współczesnym wymaganiom. Jednym z wielu jest wysoki poziom skalowalności. Relacyjne bazy danych mogą być skalowalne, zaś poziom tej skalowalności będzie ograniczony przez sprzęt. Bazy NoSQL skalowane są w poziomie, co oznacza, że sprzęt nie jest ograniczeniem, ponieważ maszyny serwerowe można połączyć, co pozwala zamienić posiadanie jednego kosztownego serwera kilkoma mniejszymi i tańszymi. Więc wydajniejsze jest użycie baz NoSQL. Użycie pamięci ulotnej w przeciwieństwie do relacyjnych baz danych, zwiększa wydajność baz NoSQL: wykonanie podstawowych zapytań, odczytu i aktualizacji. Jednakże nie zastąpią całkowicie relacyjnych baz danych. Relacyjne bazy danych są łatwe do wdrożenia, niezawodne, spójne i bezpieczne, ale nie radzą sobie z niestrukturalnymi danymi.

Trójka badaczy z Indii dokonała badania na uniwersytecie Patiala. W swoim artykule [8] przedstawili kilka prac badawczych mających na celu analizę technologii NoSQL. Zestawiono ze sobą cztery kategorie baz NoSQL: rodzina kolumn, klucz-wartość, grafowe oraz dokumentowe. Z badania wynika, że dla aplikacji wymagającej połączenia z mediami społecznościowymi, najlepszą bazą NoSQL będzie ta bazująca na grafie. Jeśli czas odpowiedzi na zapytanie powinien być jak najkrótszy, najkorzystniejszą bazą będzie typ klucz-wartość. Skalowalność jest również ważnym parametrem w przetwarzaniu dużej ilości danych. Najlepszą pod tym względem jest rodzaj: rodzina kolumn oraz klucz-wartość.

Doktor Małgorzata Plechawska-Wójcik wraz z Damianem Rykowskim z Politechniki Lubelskiej[3] przedstawili porównanie baz relacyjnych (PostgreSQL), dokumentowych (MongoDB) oraz grafowych (Neo4J). Badania przeprowadzono pod kątem wydajnościowym. Aplikacja

testowa to sieć społecznościowa. Testy wydajności przedstawiają wyniki uzyskane dla pięciu różnych zadań. A mianowicie: wybór obiektów powiązanych z przyjaciółmi, wyszukiwanie pełno-tekstowe, znalezienie liczby przyjaciół, ładowanie tematu oraz ocena tematu. Te zadania zostały wykonane dla różnych przypadków danych. A mianowicie dla każdego z 1000, 5000, 10000 przykładowych danych. Analiza wykazała, że baza grafowa najbardziej odpowiada wymaganiom danej aplikacji. Najprostsze w użyciu są zapytania do bazy dokumentowej. Pod względem wydajności PostgreSQL okazał się najlepszy.

Czwórka badaczy z uniwersytetu Jadavpur w Indiach[9] skupiła się na czterech różnych systemach baz danych: strukturalnych (PostgreSQL) i niestrukturalnych (MongoDB, OrientDB, Neo4J). Używając małego zestawu danych, na podstawie prostych zapytań wykonują różne eksperymenty, analizują i porównują wydajność wybranych baz. W eksperymencie wykorzystano zalety każdej z baz. Aplikacja testująca została opracowana wykorzystując zalety baz. Dla PostgreSQL przykładowe dane zawarte są w tabeli „Student" z kolumnami: nazwą, szkołą, klasą, wiekiem. Podobnie w przypadku MongoDB kolekcja „Student" jest reprezentowana jako pary pole-wartość zgodnie ze strukturą dokumentu JSON. Tutaj „nazwa", „szkoła", „klasa", „wiek" są reprezentowane jako klucz wraz z pewnymi wartościami. W przypadku OrientDB dane są przedstawiane jako dokument JSON. W Neo4j „Student" odpowiada nazwie etykiety, która łączy wszystkie węzły lub dane, które są tworzone. Tutaj „nazwa", „szkoła", „klasa", „wiek" to wszystkie właściwości węzła. Wydajność czterech rozważanych baz danych została oceniona na podstawie siedmiu różnych kryteriów: czas tworzenia tabeli lub zbioru, wstawienie jednego elementu danych, wstawianie N rekordów jednocześnie, zapytanie z jednym warunkiem filtrowania, zapytanie o wszystkie rekordy (całkowita liczba = 21), usunięcie jednego rekordu, usunięcie wszystkich rekordów. Z analizy wynika, że początkowy koszt instalacji Neo4j jest większy, ale koszt wyboru, wstawienia, usunięcia jest najbardziej kosztowny dla PostgreSQL. We wszystkich kategoriach zwycięża MongoDB.

3. Opis procedury badawczej

3.1. Platfora testowa

Do przeprowadzenia badań wykorzystany został komputer przenośny o następujących parametrach:

- procesor: Intel Core i5 6300HQ,
- pamięć operacyjna: 2x4 GB DDR4,
- dysk twardy: magnetyczny Toshiba MQ02ABD100H,
- system: Windows 10.

Testy zostały przeprowadzone przy minimalnym obciążeniu maszyny, gdzie w momencie każdego z testów uruchomione było tylko środowisko jednej bazy danych.

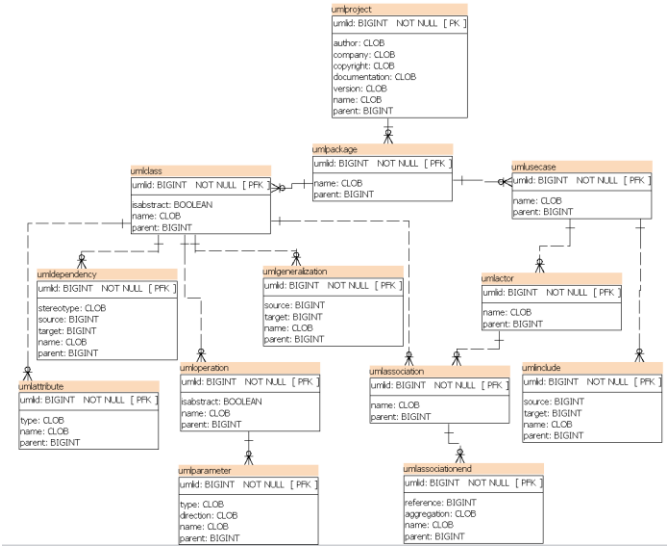
3.2. Dane badawcze

Zbiorem testowym stały się dane reprezentujące wewnętrzną strukturę projektu aplikacji StartUML, czyli dane wykorzystywane przez StartUML do zapisu projektu (diagramów UML) użytkownika na dysku [10]. Projekt taki składa się z definicji projektu, definicji widoku diagramów oraz definicji modeli z grupowaniem i podziałem na typy. Zapisany jest w pliku o rozszerzeniu *.mdj* w formacie JSON. Zawartość tego pliku, zwana dalej projektem UML, będzie wykorzystywana do zapisu i odczytu z poszczególnych baz danych.

W przypadku relacyjnej bazy danych (PostgreSQL) było wymagane stworzenie struktury tabel do przechowywania danych (Rys. 1.).

Przedstawiony diagram ukazuje jedynie główne zależności pomiędzy obiektami wykorzystanymi w badaniach. Struktura aplikacji StartUML pozwala jednak na dowolne zawieranie się elementów w elementach grupujących takich jak *UMLClass*. Z tego powodu schemat ERD nie zwiera powiązań kluczy obcych. Łączenie obiektów odbywa się poprzez identyfikatory *parent* oraz *umlid* zawarte we wszystkich obiektach. Widoczne na rysunku 1 powiązania prezentują jedynie przykładowe zależności pomiędzy tabelami, które pojawiły się w danych testowych.

UMLProject to projekt UML. Element grupujący najwyższego poziomu zawiera wszystkie informacje wymagane do wczytania diagramów stworzonych przez użytkownika w aplikacji StartUML.



Rys. 1. Poglądowy diagram ERD bazy PostgreSQL

UMLPackage to elementy grupujące odpowiadające za budowę diagramów np.: klas, przypadków użycia. Reprezentują pojedyncze diagramy i są wymagane do budowy poszczególne ekranów wyświetlających graficzny schemat modelu. W danych testowych nie zawarto elementów *UMLDiagram*, które pierwotnie znajdowały się w pakietach (*UMLPackage*) i opisywały rozmieszczenie poszczególnych obiektów graficznych, ich kolorystykę, wielkość oraz styl. Zdecydowano się na taki ruch, ponieważ wspomniane elementy nie są wymagane do poprawnej reprezentacji modelu oraz nie wpływają na jego znaczenie – nie wnoszą do

niego nic wartościowego, natomiast powodują znaczący rozrost danych reprezentujących diagramy *UMLPackage* oraz zwiększają poziom skomplikowania kodu aplikacji.

Pozostałe, nie omówione powyżej obiekty są elementami diagramów – reprezentują fragmenty ich struktury, między innymi: klasy *UMLClass*, przypadki użycia *UMLUseCase*, aktorów *UMLActor*.

3.3. Aplikacja testowa

Przeprowadzenie tak dużej liczby testów wymagało odpowiedniego skryptu lub aplikacji, więc zaimplementowana została aplikacja w języku Java. Jej zadaniem było wczytanie projektu UML i stworzenie jego reprezentacji w postaci obiektów języka Java. Tak przygotowane obiekty znacznie uprościły dalsze przetwarzanie projektu i jego wstawianie do bazy relacyjnej i grafowej, ponieważ bazy te wymagają zdefiniowanej struktury (przynajmniej na najwyższym poziomie w przypadku baz grafowych). Aplikacja umożliwiała również zapis zebranych wyników w plikach w formacie csv.

W przypadku testów wymagających więcej niż jednej instancji projektu UML, aplikacja wygenerowała pozostałe, kopiując bazowy projekt, wstawiając nowe identyfikatory oraz aktualizując powiązania.

3.4. Scenariusze badawcze

Aplikacja wykonała pomiary czasu odpowiedzi w 5 kategoriach:

- T1 – wstawianie pojedynczego projektu UML.
- T2 – wstawianie 100 unikalnych projektów.
- T3 – pobranie całego projektu UML wykorzystując identyfikator liczbowy wygenerowany poza bazą danych.
- T4 – pobranie całego projektu UML zawierającego obiekt aktor (UMLActor) wykorzystując identyfikator liczbowy aktora.
- T5– pobranie obiektów UMLClass zawierających w swojej nazwie frazę Builder.

Testy zostały przeprowadzone w kolejności wymienionej powyżej. Dodatkowo pod uwagę wzięto poziom skomplikowania kodu odpowiedzialnego za odwzorowanie projektu UML w języku Java oraz łatwość dostępu do bardziej zagnieżdżonych danych. Przypisano im jedną wartość z następujących trzech: łatwy, przeciętny, trudny.

Zbadano również poziom zajmowanej pamięci w trzech różnych momentach w czasie:

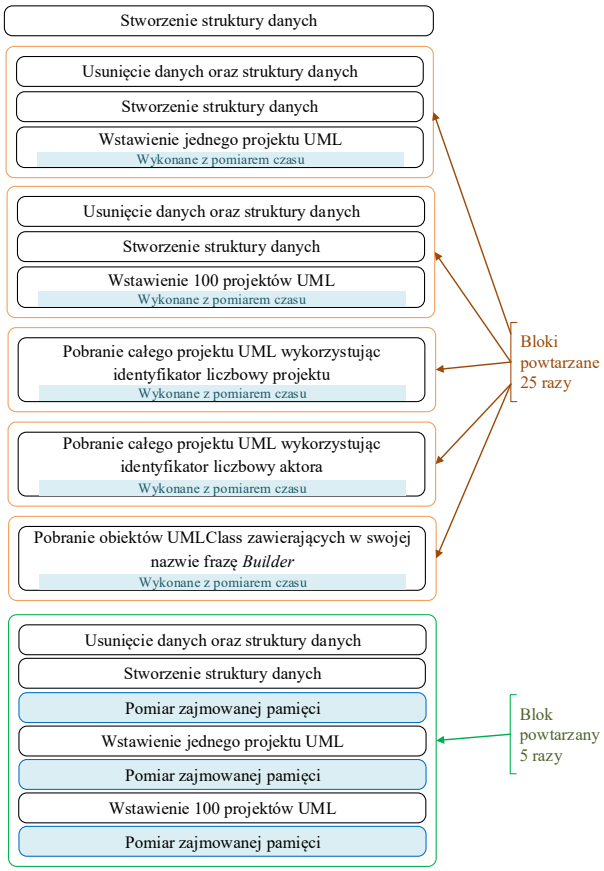
- T6 – przed wstawieniem danych,
- T7 – po wstawieniu jednego projektu,
- T8 – po wstawieniu 100 projektów.

3.5. Przebieg badania

Testy wykonywane były dla poszczególnych baz po kolei, tzn. wszystkie testy dla pierwszej bazy, następnie wszystkie dla drugiej, itd. W momencie testowania konkretnej bazy, pozostałe dwie były wyłączone aby nie wprowadzać zakłóceń w pomiarach. Rys. 2. przedstawia przebieg testów dla pojedynczej bazy.

Dla każdej bazy wykonano ten sam zestaw testów. Pierwsze 5 testówpowtarzane było po 25 razy. Powtórzenia wykonywane były dla poszczególnych testów niezależnie, tzn. wszystkie powtórzenia dla pierwszego, następnie wszystkie dla drugiego, itd. Dla pomiarów zajmowanej pamięci zdecydowano na wykonanie po 5 prób, ponieważ każda próba wymagała wyłączenia i ponownego włączenia bazy. Było to jedyne możliwe rozwiązanie problemu całkowitego wyczyszczenia danych dla bazy Neo4J z powodu alokacji pamięci przez tą bazę bez względu na ilość danych.

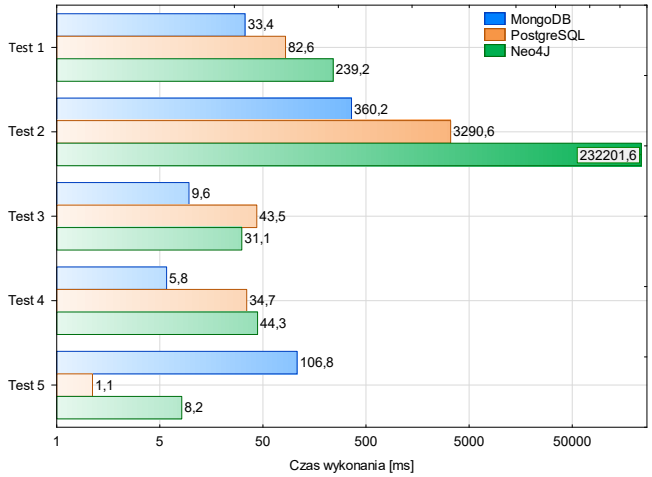
Ze względu na specyfikę baz dokumentowych, test dotyczący pobierania klas *UMLClass* dla MongoDB przebiegał nieco inaczej niż dla pozostałych baz. Ograniczeniem bazy dokumentowej była możliwość pobierania tylko dokumentów nadrzędnych bez wydzielania zagnieżdżonych, tzn. możliwe było tylko pobieranie całych projektów, a nie klas *UMLClass.* Dlatego podczas tego testu wybrano wszystkie projekty zawierające wybrane klasy, a następnie w kodzie Java wyszukano wskazane obiekty *UMLClass* i zwrócono.



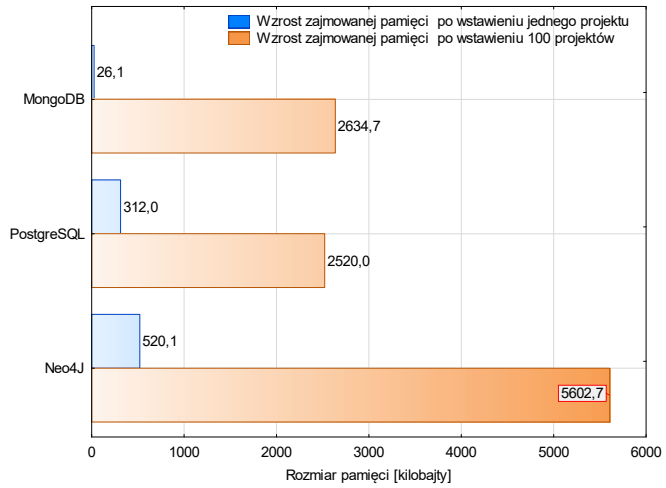
Rys. 2. Przebieg testu

4. Wyniki badania

Pomiary czasu odpowiedzi przedstawiono na rysunku 3. Rysunek 4 obrazujeróżnice w zajmowanej pamięci pomiędzy kolejnymi seriami badań.



Rys. 3. Podsumowanie wyników pomiaru czasu odpowiedzi zapytań



Rys. 4. Wzrost zajmowanej pamięci po wstawieniu 1 oraz 100 projektów

W tabeli 1 przedstawiono zaobserwowane wyniki oceny trudności w kategorii odwzorowania struktury danych w kodzie oraz dostępu do zagnieżdżonych danych. Wykorzystano trójstopniową skalę: łatwy, przeciętny, trudny.

Tabela 1. Wyniki badań trudności obsługi przetwarzania danych

	PostgreSQL	**MongoDB**	**Neo4J**
Trudność odwzorowania w kodzie	trudny	łatwy	trudny
Trudność dostępu do zagnieżdżonych danych	trudny	łatwy	łatwy

5. Dyskusja otrzymancyh wyników

W kategorii wstawiania danych, bazą o najkrótszym czasie odpowiedzi jest MongoDB. Zwycięża prawie dwukrotnie z Neo4J w teście 2. W przypadku pobierania danych, czas odpowiedzi zależy od sposobu przechowywania danych. Bazy PostgreSQL oraz Neo4J zapisują poszczególne elementy modelu (np. *UMLPackage*, *UMLCLass*) w oddzielnych strukturach danych. Dla tych baz widoczne jest zmniejszenie czasu odpowiedzi wraz z zawężeniem zakresu pobieranych danych – pobieranie samej klasy w teście nr 5 jest szybsze porównując z testami nr 3 i 4. Pomiary dla MongoDB potwierdzają wspomnianą powyżej zależność. MongoDB nie posiada oddzielnej struktury danych dla poszczególnych elementów, ponieważ jest to baza dokumentowa i przechowuje cały projekt UML w pojedynczym dokumencie. Wykazuje więc wzrost czasu dostępu do zagnieżdżonych danych ze względu na wymagane przetwarzanie danych po ich pobraniu w celu uzyskania żądanych danych. W teście nr 5 wspomniane przetwarzanie to przekształcenie danych oraz wyszukanie klas *UMLClass* o zadanej nazwie (Rys. 3.).

W przypadku pomiarów pamięci najważniejsze są nie tyle wartości bezwzględne, co różnice pomiędzy kolejnymi seriami badań, ponieważ istotna jest wielkość wzrostu pamięci wraz ze wstawianiem kolejnych danych. Im większe wartości, tym zapotrzebowanie na pamięć bardziej wzrasta (Rys. 4.).

Tempo przyrostu zajmowanej pamięci jest najwyższe dla bazy MongoDB i wynosi ok. 100-krotność (dla pozostałych ok. 10-krotność), tzn. rozmiar pamięci po wstawieniu danych jest ok. 100 razy większy niż przed ich wstawieniem. Oznacza to również, że dla każdego wstawionego projektu UML zajmowana pamięć zwiększa się o stałą wartość równą wielkości pamięci dla pierwszego projektu. Bazy PostgreSQL oraz Neo4J lepiej zarządzają pamięcią, ponieważ wraz z wstawianiem kolejnych danych przyrost pamięci zmniejsza się – wstawienie kolejnych 100 projektów UML zwiększyło zajmowaną pamięć przez te bazy tylko około 10-krotnie.

Wysoka trudność odwzorowania w kodzie dla bazy PostgreSQL oraz Neo4J wynika z wymogu stworzenia w kodzie programu pełnej struktury obiektów, które definiują poszczególne fragmenty modelu, np. przypadki użycia, klasy, aktywności(Tabela 1). Wszystkie te obiekty wymagają zdefiniowania pól dostępnych dla każdego z nich, za pomocą których program będzie tworzył żądania do bazy danych. Każdy nowy fragment modelu o unikatowych właściwościach wymaga utworzenia unikatowego dla tego obiektu odwzorowania. Natomiast dla bazy MongoDB utworzenia odwzorowania jest opcjonalne, ponieważ akceptuje ona projekt w postaci mapy tj. kolekcji klucz – wartość. Odwzorowanie może być również częściowe. W MongoDB przekształcanie danych oraz ich przekazanie do bazy jest mniej skomplikowane, a cały projekt może być zapisany lub odczytany przy pomocy jednego żądania.

Dostęp do zagnieżdżonych danych w bazie PostgreSQL jest dużo trudniejszy niż w dwóch pozostałych, ponieważ

wymaga tworzenia rozbudowanych żądań zawierających wiele zależności. Kolejne żądania wymagają identyfikatorów elementów nadrzędnych, aby możliwe było ich wykonanie. Powoduje to dłuższy czas oczekiwania na dane oraz bardziej złożoną implementację. W MongoDB dostęp do danych zawieranych przez obiekt jest prostszy – odbywa się za pomocą znaku kropki („.”), a w Neo4J na podobnej zasadzie – przy użyciu symbolu relacji („-->” lub „-[nazwa]->”).

6. Podsumowanie i wnioski

Celem niniejszej pracy była analiza porównawcza efektywności składowania oraz dostępu do modeli UML na przykładzie wybranych technologii bazodanowych. Przeprowadzone analizy potwierdzają prawdziwość hipotezy, iż baza dokumentowa reprezentowana przez MongoDB jest najlepszym wyborem w kwestii efektywnej obsługi struktur danych modeli UML. Technologia MongoDB zwycięża w 4 z 5 testów wydajności. Jedynie pobieranie danych zagnieżdżonych powoduje pewne problemy i wymaga mocy obliczeniowej aplikacji testowej. Jednak problem ten w większości sytuacji może być zminimalizowany poprzez przeniesienie procesu przetwarzania na stronę serwerową posiadającą znacznie większe zasoby sprzętowe.

Dodatkowo baza MongoDB jest najlepszym wyborem ze względu na trudność odwzorowania struktury danych modeli UML w kodzie aplikacji – pozwala na stworzenie aplikacji nie zwierającej takiego odwzorowania, a sama baza go nie wymaga. Poza tym dostęp do zagnieżdżonych danych, podczas tworzenia zapytań, jest wśród wszystkich testowanych technologii najłatwiejszy.

W kategorii wielkości zajmowanej pamięci baza MongoDB odznacza się najgorszym wynikiem na tle dwóch pozostałych. Nie wykazała ona widocznych optymalizacji rozrostu pamięci i pomimo małego rozmiaru początkowego już dla 100 wstawionych projektów zużycie pamięci zrównuje się z bazą PostgreSQL przechowującą tę samą liczbę projektów. Należy zaznaczyć, że otrzymany współczynnik przyrostu pamięci jest liniowy, dlatego nie można powiedzieć, że baza MongoDB jest nieodpowiednia do przechowywania danych struktur modeli UML, jednak należy zwrócić uwagę na zapewnienie jej odpowiednio dużej przestrzeni dyskowej, co ułatwia jej wysoki poziom skalowalności w poziomie.

Test nr 5 pokazuje, że technologia baz dokumentowych reprezentowanych przez MongoDB, nie radzi sobie dobrze z pobieraniem obiektów zagnieżdżonych – wymaga pobrania całego dokumentu oraz późniejszego jego filtrowania. Z tego powodu wraz ze wzrostem poziomu zagnieżdżeń obiektów danych (na poziom o wiele wyższy niż wykorzystany w testowym projekcie UML), wartą rozważenia będzie baza Neo4J ze względu na łatwy sposób budowania zapytań oraz uzyskane korzystne pomiary.

Literatura

[1] The Coming Software Apocalypse. https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/ [05.05.2019].

[2] Jung M., Youn S., Bae J. Choi Y.: A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment. 8th International Conference on Database Theory and Application (DTA). IEEE, 2015.

[3] Plechawska-Wójcik M., Rykowski, D.: Comparison of relational, document and graph databases in the context of the web application development. 36th ISAT Conference, Part II, s. 3-13. Springer, Cham, 2015.

[4] Van der Veen J. S., Van der Waaij B., Meijer R. J.: Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. Fifth International Conference on Cloud Computing, Honolulu, s. 431-438. IEEE, 2012.

[5] Oussous A., Benjelloun F. Z., Lahcen A. A., Belfkih S.: Comparison and classification of nosql databases for big data. International Journal of Database Theory and Application, 6, April, 2013.

[6] Shetty Deepika V., Chidimar S. J.: Comparative Study of SQL and NoSQL Databases to evaluate their suitability for Big Data Application. International Journal of Computer Science and Information Technology Research, June, s. 314-318. 2016.

[7] Kunda, D., Phiri, H.: A Comparative Study of NoSQL and Relational Database. Zambia ICT Journal. Tom 1, s. 1-4. 2017

[8] Bathla, G., Rani, R., Aggarwal, H.: Comparative study of NoSQL databases for big data storage. International Journal of Engineering & Technology, 2018.

[9] Mondal, A. S., Sanyal, M., Chattopadhyay, S., Mondal, K. C.: Comparative Analysis of Structured and Un-Structured Databases. International Conference on Computational Intelligence, Communications, and Business Analytics, s. 226-241. Springer, Singapore, March, 2017.

[10] Introduction - StarUML documentation. https://docs.staruml.io/[05.05.2019].